Sustainable CSE Software Engineering and Support: Lessons Learned from the Trilinos Project

Michael A. Heroux*, Sandia National Laboratories
April 2009

Introduction

The Trilinos Project is nearly ten years old and has grown from the initial "grand" plan of three equation solver packages (thus the "tri" in Trilinos) to now contain approximately 50 packages whose functionality covers much of the enabling capabilities needed to construct a scalable computational science and engineering (CSE) application. Trilinos is an open source project with more than 3,500 registered users worldwide, 2,000 of which are at universities. Trilinos is the scalable library foundation for most applications at Sandia National Laboratories and is increasingly used in the same way by many other CSE applications.

From the beginning of the Trilinos Project, we have invested in software engineering practices and tools with the intent to elevate the quality and maintainability of our software for the long-term future. Our guiding principle is this:
We carefully invest in software engineering so that we can spend less time writing and maintaining software, and more time computing science and engineering results.

Our approach is pragmatic and measured, realizing that we need to continue CSE research while at the same time gradually improving our software engineering efforts.

General Observations

The Trilinos Project grew out of several observations about existing CSE software efforts at Sandia and elsewhere. Specifically:

1. **Small teams are natural:** Single-physics CSE applications and library software efforts naturally involve a small team of researchers who work closely with each other on a daily basis.
2. **Unneeded redundancy is a risk:** These small team efforts often require—and redundantly develop if left alone—similar basic functionality. This basic functionality can come in the form of I/O functions, interfaces to basic libraries or the creation in one library of an inferior service capability, e.g., linear solvers, which are available as state of the art

* Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

capabilities in another library that should have been leveraged.

3. **Large-scale projects are compositions of small teams:** Advanced CSE projects require a coordinated effort of dozens or more researchers who, although contributing to a larger effort, continue to work in small teams on their portion of the project.
4. **Advanced research needs advanced software:** Pre-existing, high-quality software is increasingly required in order to perform advanced research. For example, to perform research in large-scale optimization or uncertainty quantification we need scalable data classes and scalable linear and nonlinear solvers, and all of these capabilities must work together seamlessly.

A Federalist Approach

The Trilinos project uses a kind of "federalist" approach to addressing these competing realities. We have formally defined a "package" to be a collection of related functionality developed by a small team with certain rights and responsibilities in the larger Trilinos framework.

This basic approach has enabled a great deal of local autonomy in decision-making, allowing us to tolerate and appreciate a variety software engineering styles and team cultures. We can handle modest redundancy in software functionality (that is gradually eliminated based on the merit of competing approaches) and can adapt to change in many ways. At the same time, this approach also provides a global interaction that promotes a variety of desirable outcomes: (i) cross-fertilization of ideas, techniques and tools across package teams, (ii) adoption of "best practices" from one package across other packages, (iii) fostering of trust among disparate groups (iv) software modularity that is naturally enforced by package and team boundaries and (v) well-defined interfaces between packages for interoperability.

Importance of Software Excellence Emphasis

One important factor that improves the effectiveness of the Trilinos architecture is the constant focus on improving software engineering practices and processes. This focus has two major impacts on our efforts: (i) better software engineering in the project makes for better software, so that package teams are willing to use each other's software and (ii) discussions of incompatibilities in practices and processes across packages can focus on the goal of determining best practices and not decay into expressions of personal preference that can be contentious and counter-productive. It is worth noting, that we selectively adopt practices from the larger software engineering community only when deemed to be useful to us, doing so in a timely manner.

The net result of this approach to software development is a large and growing collection of inter-related tools where Trilinos as a whole has an identity but, even more importantly, each package has its own identity within its community of interest. Furthermore, even though our software base has grown extensively, our overall support costs have remained essentially constant because of the payoffs from better software engineering practices.

On-going Software Support

Perhaps the most challenging task in CSE software development is the maintenance of existing software. In our experience, maintenance challenges are of two types:

1. **Maintenance of actively developed software:** In this situation, maintenance is often easy to perform and fund because development due to maintenance issues, and development related to new software features can be done simultaneously. As long a software project is actively developed, we have not found maintenance a major problem. However, proper support staffing (see below) is needed.
2. **Maintenance of mature "legacy" software:** Mature legacy software that is popular and used but minimally developed, or not actively developed at all, is the most challenging type to maintain. Particularly challenging is the situation where minor refactoring is required—say to support new architectures or non-research software features—that in turn requires ubiquitous changes. Research funding cannot typically be used to perform this maintenance so it either is not done, or is done for a commercial customer who pays for the work and is therefore the owner of the best version of software, leaving the open community with a lower quality version.

Persistent CSE Software Support Staffing

Most CSE software development is done by highly trained domain scientists who have to do some amount of low-level programming as part of their work. At the same time, we have found that support staff members are essential for a sustainable software project. These members of the team should be long-term project members, providing continuity even if research staff members such as students and post-docs depart.

There are two basic support staff roles:

1. **Software Engineering Tools Support:** Responsible for use and support of software tools such as source management, issue tracking, websites, etc.
2. **Technical Software Development:** Responsible for software development that is needed to provide software refactoring of a technical nature, such as introducing a new parallel programming approach into existing software.

We have found it easier to justify the first type of support staffing than the second, but they are equally important.

Increasing the Priority of Software Engineering and Support

In our experience, the broad CSE community is largely unaware of formal software engineering concepts and practices, and in fact has a negative opinion if any. The Trilinos Project team would probably have a similar attitude but for the fact that some of our important funding is explicitly connected to demonstrating software quality and participating in software quality assessments. Simultaneously these same funding sources provide resources for focusing on software engineering, allowing us to hire dedicated personnel who are primarily focused on software engineering and not on CSE research.

Finally, it is *very* important to carefully incorporate software engineering principles and practices into CSE software efforts. In particular, we have found that blind adoption of standard industry practices is not a good approach and is largely responsible for the existing negative opinion of formal software engineering in the CSE community. Furthermore, it is very important to provide educational opportunities for CSE researchers to learn about basic software engineering concepts and to support ongoing educational opportunities.

Conclusions

Traditionally CSE software engineering and support efforts have been considered to be outside the scope of CSE research funding. However, in our experience these efforts are essential elements of a full-fledged CSE research program. They are not overhead or luxuries, but instead free up research staff to do more research by providing software engineering expertise to domain scientists, and ultimately allowing highly trained domain scientists to be more focused on CSE research. Furthermore, some areas of CSE research fundamentally require high quality CSE software upon which to build. These research areas are off-limits to CSE researchers without a stable and supported CSE software base. Finally, an increased focus on software engineering and support requires explicit funding and staffing, and requires education of CSE researchers in software engineering topics.